

# **Design and Implementation of A Smart Grid System Based on Blockchain Smart Contract Technology**

Xueyuan Fu

Thesis - Examination  
Main field of study: Computer Engineering  
Credits: 15  
Semester/Year: Autumn 2020  
Supervisor: Stefan Forsström  
Examiner: Patrik Österberg  
Course code/registration number: DT099G

# Abstract

In recent years, blockchain technology has received more and more attention. It has shown special advantages in digital currency, because it is distributed and its data cannot be altered.

With more countries put forward the idea of the power system innovation, a large number of distributed power sources have emerged. The grid connection of these distributed power sources will lead to unstable grid operation and greatly increase the difficulty of management. Therefore, there is an urgent need for a solution that can realize direct transaction of distributed power supply.

This article makes an in-depth analysis of the blockchain technology, including hash algorithm, consensus mechanism, Merkle tree, smart contract, etc. And then the Ethereum and smart grids are studied. This article realizes automation and intelligence of the electricity transaction measurement through the smart contract technology provided by Ethereum. A blockchain private chain is created and then the smart contract is deployed into the private chain. With the advantages of blockchain technology aforementioned, the storage of power data and the power transactions will be more credible and more transparent.

All in all, this paper designs and builds a smart grid system based on the smart contract technology of blockchain. The system can be used not only for smart grid systems but also for other energy trading systems. This article provides a reference for the application of blockchain technology.

**Keywords:** Blockchain; Ethereum; Smart contracts; Smart Grid; Hash Algorithm

# Sammanfattning

Under de senaste åren har blockchain-tekniken fått mer och mer uppmärksamhet. Det har visat speciella fördelar i digital valuta, eftersom det distribueras och dess data inte kan ändras.

Med fler länder som presenterar idén om kraftsysteminnovationen har ett stort antal distribuerade kraftkällor dykt upp. Nätanslutningen för dessa distribuerade kraftkällor leder till instabil nätdrift och ökar svårigheten att hantera kraftigt. Därför finns det ett akut behov av en lösning som kan realisera direkt transaktion av distribuerad kraftförsörjning.

Denna artikel gör en fördjupad analys av blockchain-tekniken, inklusive hasalgoritmer, konsensusmekanism, Merkle-träd, smart kontrakt etc. Och sedan studeras Ethereum och smarta nät. Den här artikeln realiserar automatisering och intelligens för mätning av eltransaktioner genom smart kontraktsteknik som tillhandahålls av Ethereum. En blockchain privat kedja skapas och sedan distribueras det smarta kontraktet i den privata kedjan. Med fördelarna med blockchain-tekniken ovan kommer lagring av kraftdata och krafttransaktioner att vara mer trovärdig och mer transparent.

Sammantaget designar och bygger detta papper ett smart grid-system baserat på blockchain-smarta teknik. Systemet kan inte bara användas för smarta nätsystem utan även för andra energisystem. Denna artikel ger en referens för tillämpning av blockchain-teknik.

**Nyckelord:** Blockchain; Ethereum; Smarta kontrakt; Smarta elnät; Hash-algoritm

# Table of Contents

<b>1</b>	<b>Introduction .....</b>	<b>1</b>
1.1	Background and problem motivation.....	1
1.2	Overall aim.....	2
1.3	Concrete and verifiable goals .....	3
1.4	Outline .....	3
<b>2</b>	<b>Theory .....</b>	<b>5</b>
2.1	Blockchain .....	5
2.1.1	Block .....	5
2.1.2	Block header .....	6
2.1.3	Longest chain .....	6
2.1.4	Bookkeeping right .....	6
2.1.5	Consensus mechanism.....	7
2.1.6	Hash algorithm .....	7
2.1.7	Merkle tree.....	7
2.1.8	Smart contract .....	8
2.2	Ethereum .....	8
2.3	Smart Grids .....	9
<b>3</b>	<b>Methodology .....</b>	<b>11</b>
<b>4</b>	<b>Design .....</b>	<b>12</b>
4.1	Overall process of smart contract .....	12
4.2	Construction of private chain based on Ethereum .....	13
4.3	Deployment of smart contract on private chain.....	13
<b>5</b>	<b>Implementation .....</b>	<b>14</b>
5.1	Smart contract Writing.....	14
5.1.1	Creation and initialization of related member variables .....	14
5.1.2	Implementation of related member function .....	16
5.2	Compilation and debugging of smart contract .....	20
5.2.1	Compilation of smart contract.....	20
5.2.2	Debugging and running of smart contract .....	22
5.3	Build a private chain.....	26
5.4	Deploy smart contract .....	29
5.4.1	Initialize truffle .....	29
5.4.2	Compile and migrate smart contract .....	30
5.5	Summary of this chapter .....	31
<b>6</b>	<b>Results .....</b>	<b>32</b>
6.1	Fixed number of users, different number of blocks.....	32
6.2	Different number of users, fixed number of blocks .....	33

6.3	Analysis of results.....	34
7	<b>Conclusions</b> .....	<b>36</b>
7.1	Ethical and social aspects.....	37
7.2	Future work .....	37

# 1 Introduction

Now we are heading towards a future where more households have their own electricity production. In Sweden there was for example a production capacity of approximately 127 MW from all installed in solar cells in 2017 [1] . In 2016, as many as 21.5% of all new cars in Sweden were electric or hybrid cars [2] . Which means that there are many households that own cars with large batteries that for the most time stays unused in their garages. We can also see a future with smart grids[3] , where power grids become more than just a static infrastructure and supplier of electricity. The term smart grid includes wide research areas from power electronics and new technology in the hardware grid, to new products and services based on the data and information available in the system. The smart grid can therefore be seen as a next generation power grid, which uses two-way flows of electricity and information to create a distributed automated energy delivery system [4] .

## 1.1 Background and problem motivation

Blockchain technology originated from bitcoin, and bitcoin is the most successful blockchain application at present. Since the first bitcoin was produced on January 3, 2009, bitcoin system has been running continuously for more than 11 years without management of any central organization. Although its performance and security need to be improved, and the problem of high resource consumption needs further development of related technologies. However, its achievements in the field of digital currency have been obvious.

Since 2014, the blockchain technology behind bitcoin began to catch people's attention. People realized that blockchain technology is a good solution to solve the problem of trust between both sides without management of central organization. At present, blockchain technology has been independent of bitcoin, and has been applied in many fields including finance, education, medical treatment, credit investigation, Internet of things, etc[5] [6] .

Without the need of third-party trust organizations, blockchain have the characteristics of both decentralization and data unforgeability[7] . Therefore, blockchain technology can be applied to nearly all activities that require third-party trust guarantee organizations. Smart grid is one of them.

With the concept of energy Internet put forward, there is an urgent need for a technology that can quickly solve the problem of local consumption of distributed energy such as distributed photovoltaic, battery energy storage and electric vehicles, so as to form an energy sharing network[8]. Blockchain technology can provide solutions for the construction of energy Internet system, which can be summarized as follows.

- 1) Credible measurement. The data of blockchain itself cannot be tampered with, which makes the measurement more credible, and solves the problems of high data acquisition cost in traditional power management.
- 2) Intelligent control. The existing smart meters combine with the blockchain smart contract technology to make the system control more intelligent, and issue various control commands through the blockchain, making the process more credible.
- 3) Cluster intelligence. The distributed characteristics of blockchain technology can coordinate different energy sources and make the energy Internet more intelligent.

Therefore, while paying attention to the upsurge of blockchain technology, it is necessary to study the blockchain technology deeply and study its application in various fields by using the characteristics of blockchain technology. In the time of energy shortage, when renewable energy plays an increasingly important role, the use of blockchain technology to accelerate the construction of energy Internet is particularly important. This paper takes the smart grid as a representative to provide ideas for the application of blockchain technology in the energy Internet.

## **1.2 Overall aim**

Firstly, we study the underlying technology of blockchain. Blockchain technology is actually a collection of some existing technologies, including cryptography, P2P, distributed database and so on. Among them, cryptography runs through the whole blockchain system, including hash algorithm and asymmetric encryption algorithm; distributed consensus mechanism mainly includes P2P and workload proof mechanism. Next, we introduce the working principle of blockchain, understand Merkle tree, and introduce the structure of

blockchain, including block header and block body. Finally, we introduces Ethereum. Ethereum is equivalent to encapsulating the blockchain. Its Turing complete script running system provides a more convenient blockchain development platform. Based on Intelligent contract technology, it reduces the threshold for users to build blockchain applications.

Secondly, by writing smart contracts and deploying smart contracts in the private chain of blockchain, the basic function of the smart grid trading platform is realized, so that the power transaction data can be recorded in the blockchain to ensure the safety and reliability of the data.

### **1.3 Concrete and verifiable goals**

With the system we want to build, this paper has the following concrete goals:

- 1) Survey the area of blockchains for smart grids, as well as energy trading.
- 2) Test the accuracy of smart contracts in the test environment provided by Ethereum.
- 3) Implement the work as an open source project. Build a smart grid system based on the smart contract technology of blockchain using Ethereum.
- 4) Evaluate the end results and propose future work.

### **1.4 Outline**

The chapters of this paper are arranged as follows:

The second chapter introduces the underlying technology of blockchain, and understands the working principle of blockchain and Ethereum smart contract.

The third chapter explains the methodology used in the study of smart grid systems. Including how to write smart contracts, how to transplant smart contracts to the Ethereum private chain to form a smart grid system, and how to evaluate the performance of the system



The fourth chapter includes the overall process of the construction of private chain based on Ethereum and the deployment of smart contract.

The fifth chapter is about the compilation of smart contract and the construction of private chain. In this chapter, the smart contract is written, which realizes the basic functions of power transaction and can record the user data to the blockchain. Then, we build Private chain based on Ethereum, and deploy the smart contract to the private chain.

The sixth chapter mainly evaluates the system performance. The performance of the system was evaluated by recording the response time of the system under different loads

The seventh chapter mainly summarizes the current work and shortcomings, and makes a prospect for the next step of the research.

## 2 Theory

The following chapter will present theory about this work.

### 2.1 Blockchain

A blockchain is a long chain that is stored in a linked list structure and connected end to end in chronological order, as shown in Figure 2-1. Each block records the transfer information in the network within a time period. When a user needs to transfer the "funds" in his account to other accounts, the user will broadcast the encrypted information to the whole network through P2P network. Each node of the whole network will verify the information and store it in the trading pool correctly. The node that obtains the accounting right will package the transaction in this period into the new area block and broadcast it to the whole network[8] .



**Figure 2-1: A simple structure of blockchain**

After receiving the block, each node verifies whether the block is legal and whether the information in the block is correct. After the verification is passed, each node extends the block to its own stored blockchain, and strives for bookkeeping rights on the basis of the new blockchain to obtain new rewards. In this way, the blockchain system maintains the consistency of the whole distributed peer-to-peer network. Although the nodes in the whole network are asynchronous, they follow the setting of the longest blockchain, so that after several blocks, the blockchain of each node in the whole network can gradually converge to the same blockchain.

The basic concepts of blockchain technology are introduced as follows:

#### 2.1.1 Block

Block is a unique data unit of blockchain, which is composed of block head and data after block head.

### 2.1.2 Block header

Block header identifies some important parameters of a block, mainly including three parts: first, it refers to the hash value of the previous block, which is used to link with the original blockchain; second, some parameters, such as block difficulty, timestamp, and random value found by computational power; third, the tree root of Merkle tree of all transaction data in this block, which is used to summarize all data information in this block[9] . Take the block header of bitcoin blockchain as an example, as shown in Table 2-1.

Size	Field	Description
4 bytes	Version	The Bitcoin Version Number
32 bytes	Previous Block Hash	The previous block header hash
32 bytes	Merkle Root	A hash of the root of the merkle tree of this block's transactions
4 bytes	Timestamp	The timestamp of the block in UNIX.
4 bytes	Difficulty Target	The difficulty target for the block.
4 bytes	Nonce	The counter used by miners to generate a correct hash.

**Table 2-1: The block header of Bitcoin [10]**

### 2.1.3 Longest chain

Due to network delay and other reasons, all nodes in the network receive different blockchains. Each node will be extended on the basis of its own longest chain. Different lengths of blockchains will compete with each other. The longest blockchain is considered to be the only legal one by the network, and the remaining forks will be discarded. Therefore, the node will find the longest chain to work, and extending on the shorter chain will not be able to obtain revenue because it cannot be recognized by the network, thus wasting the cost of computing power[11] .

### 2.1.4 Bookkeeping right

The data of distributed ledger is maintained by the whole network, and the generation of new data needs to be generated by bookkeeper. Taking bitcoin as an example, the process of mining is actually all the computing power of the whole network competing for bookkeeping rights, and the

winner will get the rights to generate new blocks and get rewards. The mode of competitive bookkeeping right is determined by the type of consensus mechanism.

#### **2.1.5 Consensus mechanism**

Consensus mechanism is a solution to the Byzantine general problem in distributed peer-to-peer networks. The purpose of consensus mechanism is to make the nodes of distributed peer-to-peer networks reach an agreement. For Byzantine fault-tolerant system, the security level is quantified as the maximum proportion of malicious nodes allowed. Bitcoin adopts pow {proof of work} mechanism, which requires that the computing resources occupied by malicious nodes in the network should not exceed the normal nodes.

#### **2.1.6 Hash algorithm**

As another important encryption algorithm in bitcoin blockchain, hash algorithm is an important tool for modern information system encryption[20] . Hash calculation is actually a digest calculation. Data with different length can be compressed and mapped to a unique value of fixed size by hash algorithm. For different input X, it is possible to get the same hash value Y, but Y cannot be used to deduce the input X. By changing only one digit of X, the value of Y 'is completely different from that of Y. The transaction data in the blockchain has the corresponding hash value through the encryption of hash algorithm. Once the data is modified, the hash value will change completely. Forgery data needs to forge hash value together, and the cost of forging a single hash value is very high. The block chain further improves the difficulty of hash value forgery through Merkle tree[12] .

#### **2.1.7 Merkle tree**

Merkle tree is a binary tree containing hash encryption, used to summarize all transaction data in a block[13] . The data within each block is organized in the form of Merkle tree, as shown in Figure 2-2. From leaf to root, hash encryption is performed layer by layer to get a unique hash value. Merkle tree ensures that a series of hash values need to be modified to modify a single data.

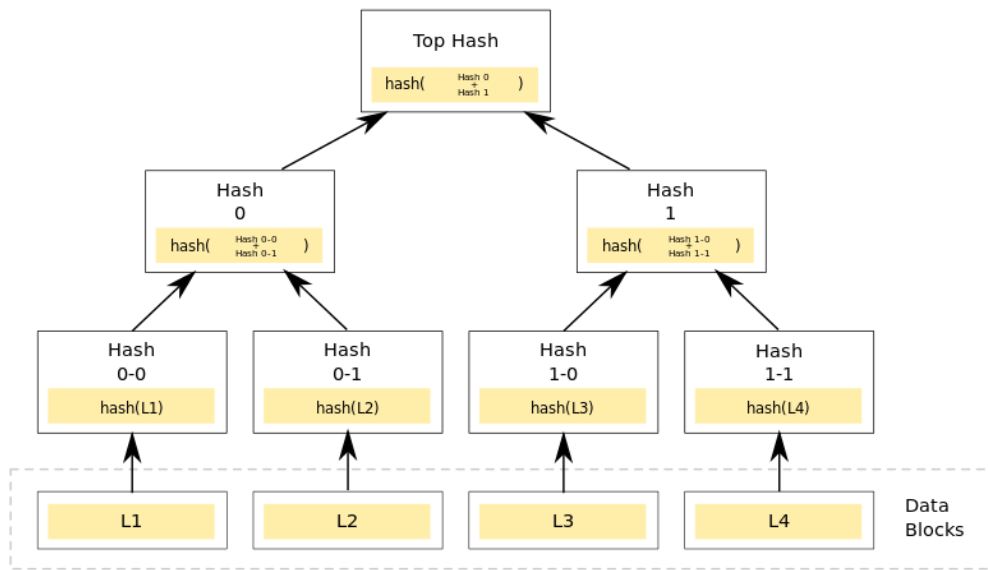


Figure 2-2: Data hashed in Merkle tree [14]

### 2.1.8 Smart contract

Smart contract is a self-enforcing piece of protocol that is managed by a P2P network of computers. Smart contracts are effective rights management tools that provide a coordination and enforcement framework for agreements between network participants, without the need of traditional legal contracts. They can be used to formalize simple agreements between two parties, the bylaws of an organization, or to create tokens[15] .

## 2.2 Ethereum

Ethereum is a blockchain development platform that supports smart contracts and reduces the threshold for users to build blockchain applications. Like bitcoin, Ethereum is an open-source blockchain underlying system. Based on bitcoin, Ethereum encapsulates the underlying blockchain technology, just like the operating system, provides a very rich external interface, enabling people to quickly develop various blockchain applications without in-depth understanding of the underlying blockchain technology. In addition, one of the biggest features of Ethereum is its combination with smart contracts. The blockchain technology provides a credible execution environment for the operation of smart contracts. It can be said that the

blockchain technology has revived the smart contract technology, and the smart contract technology has also expanded the application scope of the blockchain technology[16] .

Remix is the official ide of Ethereum. When users write code on the compiler, they can use the simulator provided by Remix to simulate the operation of smart contract, and provide contract interface, which reduces the difficulty of developers' development. In addition, the smart contract also specially developed solidity programming language, similar to python, which is relatively simple and easy to understand. After solidity language is compiled, it can convert the smart contract code into the bytecode recognized by the Ethereum virtual machine, and then publish it on the Ethereum blockchain. In addition, if you want to interact with the Ethereum node, the Ethereum client provides a variety of methods. The console interaction is the most direct way. If you want to interact with the Ethereum node in the program, you can use web3.js and JSON-RPC.

## **2.3 Smart Grids**

A smart grid is an electricity network enabling a two-way flow of electricity and data with digital communications technology enabling to detect, react and pro-act to changes in usage and multiple issues. Smart grids have self-healing capabilities and enable electricity customers to become active participants.

A smart grid serves several purposes and the movement from traditional electric grids to smart grids is driven by multiple factors, including the deregulation of the energy market, evolutions in metering, changes on the level of electricity production, decentralization (distributed energy), the advent of the involved 'prosumer', changing regulations, the rise of microgeneration and (isolated) microgrids, renewable energy mandates with more energy sources and new points where and purposes for which electricity is needed (e.g. electrical vehicle charging points)[17] .

The main difference between the Smart Grid and Conventional Power Grid is that Smart Grid is decentralized, which has many small power producers comparing with centralized large power plants in conventional power grid. And also the two-way flow of electricity and data that is the essential characteristic of a smart grid enables to feed information and data to the various stakeholders in the electricity market

which can be analyzed to optimize the grid, foresee potential issues, react faster when challenges arise and build new capacities – and services – as the power landscape is changing. The comparison between smart grid and traditional grid is shown in Figure 2-3.

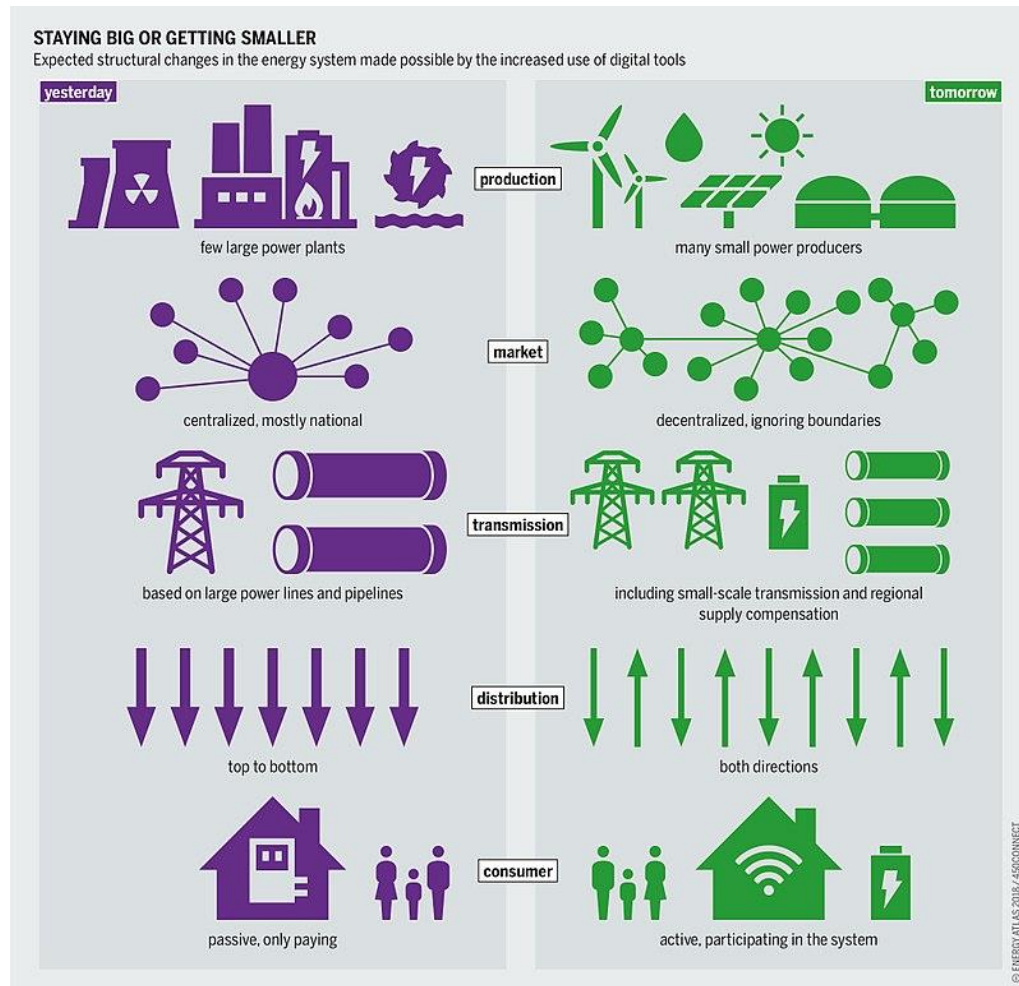


Figure 2-3: Differences between smart grid and traditional grid [18]

### 3 Methodology

This paper aims to build a smart grid system based on smart contract technology of blockchain, so first we need to write smart contracts according to the functions the system supposed to achieve. Then we need to select a suitable compilation environment to compile the smart contract and check whether it can correctly implement various functions. Then we need to build a blockchain to deploy smart contracts and realize the functions of a smart grid. Finally, we need to evaluate the performance of the system.

First of all, we should design the functions that the system should implement, and write corresponding functions in the smart contract for each function to implement it. First, the administrator can add or delete members and publish some information. Then power generators can generate electricity, and power consumers can consume electricity. They can all inquire about electric energy, and they can also realize the transaction of electric energy.

Then we have to choose a suitable environment to debug the smart contract. Here I use Remix. It is a virtual machine that simulates the blockchain in memory and is provided by Ethereum. The reason for choosing it is that it is easy to operate. A blockchain can be built by setting data on the front end of the web page to debug smart contracts.

Then we need to build a blockchain to realize the smart grid system. First of all, we should build a private chain, because it is convenient for us to manage and does not need to use virtual currency for transactions. Then we choose the truffle framework of Ethereum to implement the private chain.

At last, we need to evaluate the performance of the system. The response time of the system under different loads can reflect system performance to a certain extent. We can test the response time of the system to achieve various functions under different independent variables. The first type of test can fix the number of users and change the number of blocks. The second type of test can fix the number of blocks and change the number of users.



## 4 Design

The main content of this chapter includes the overall process of smart contract, and the construction of private chain based on Ethereum and the deployment of smart contract.

### 4.1 Overall process of smart contract

As the only administrator, the power management organization manages the power consumption and power generation rights of each user, and sends dispatching information according to the real-time operation of the power grid. Power generators can generate electricity after being authorized to generate electricity, and upload real-time generation data to the blockchain through the power generation contract interface. Power consumers can use electricity after being authorized to use electricity. The electricity is purchased from the power plant through the power purchase interface provided by the contract. After purchasing the electricity, the user will upload the real-time power consumption data to the blockchain through the power consumption interface of the contract. In addition, according to the real-time operation of the power grid, the power management organization can release the dispatching information through the dispatching information interface of the contract. So far, the contract has realized the basic functions of power generation, power sales, power consumption and dispatching information release. All the above operations will be permanently recorded in the blockchain to ensure data security and transparency of power dispatching. The overall process is shown in Figure 4-1.

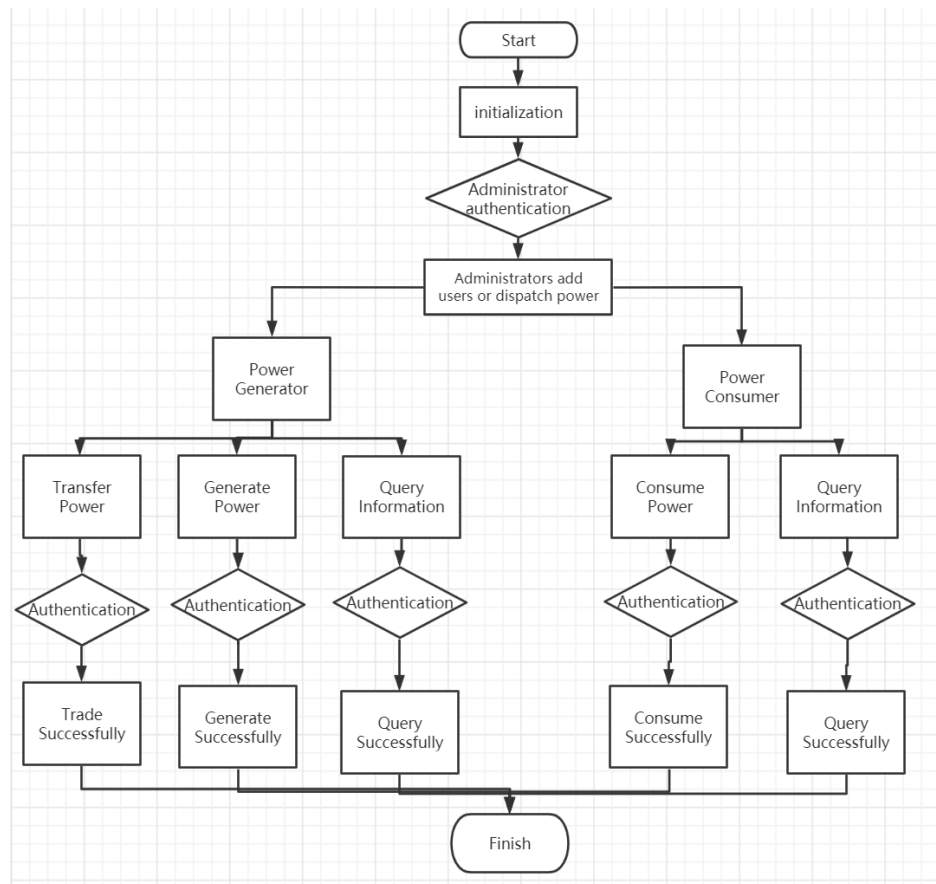


Figure 4-1: Overall process of smart contract

## 4.2 Construction of private chain based on Ethereum

Although Ethereum is a public chain system, we can run our own private chain nodes by setting some parameters. The private chain will not synchronize the data of the Ethereum public network. It does not need to spend money on buying ETH. It saves storage space and costs, and is easy to manage.

## 4.3 Deployment of smart contract on private chain

There are many ways to deploy smart contracts. Here we use the truffle framework. Truffle is one of the most popular development frameworks in Ethereum. It provides a contract abstract interface, which can be directly accessed through `var meta = PowerGeneration.deployed()`; to get the contract object. After getting the contract object, we can directly operate the corresponding contract function in JavaScript. Based on the framework of truffle, we can easily build the distributed application of Ethereum.

## 5 Implementation

The main work of this chapter includes: write smart contract, build Ethereum blockchain private chain, and deploy the contract to the blockchain. These work aims to realize the automation and intelligence of power transaction, and ensure the security of power data and the transparency of power dispatching.

### 5.1 Smart contract Writing

In this paper, Solidity language is used to write smart contract. Solidity is a high-level language with syntax similar to JavaScript. The compiler uses Remix, a browser based compiler. This compiler also provides contract debugging function. The contract code writing and side testing in the later part of this paper are based on this compiler. As the smart contract is the main part of this paper, this chapter will give the relevant code in the smart contract and explain it.

#### 5.1.1 Creation and initialization of related member variables

The smart contract only needs to be deployed once, and it can exist in the blockchain forever. After that, the contract can be executed according to the agreement. Therefore, the contract should be deployed by the power management organization, and as the only administrator, it can manage the generation and consumption units and release dispatching information. The definition and initialization of the administrator are as follows:

```
1 address private administrator;  
2 function PowerGeneration() public  
3 {  
4     administartor = msg.sender;  
5 }
```

Here, an *address* type variable is defined as the administrator address, and it is initialized to be the contract creator's address through the constructor when the contract is first deployed, and cannot be modified later. *Powergeneration* is the constructor, that is, the function with the same name as the contract class. The concept of constructor is the same as that in C + +, Java and other high-level languages.

*address* is a special type provided by solidity, which is essentially a 20 bytes Ethereum address. *msg* is a global variable provided by Solidity,

and the address of the external caller of the current function can be obtained by its member value *sender*.

The user structure is defined below, which contains the basic information of the user, such as the user name, address, and other information, as well as the user's initial total power consumption, total power generation, power consumption right, generation right, etc., and also includes a dispatching information. The power management organization can send the power dispatching information to the user through this variable. The specific definitions are as follows:

```
struct User
{
    string name;
    string useraddr;
    int256 generationPower;
    uint256 totalGenerationPower;
    uint256 usePower;
    uint256 totalUsePower;
    bool generaterights;
    bool userights;
    string information;
}
mapping(address => User) user;
```

Finally, an array container is declared, which maps the Ethereum address to the user, and each Ethereum address corresponds to a unique user structure.

Ordinary users can register as consumption users or generation users in the contract publisher, i.e. the power management agency. The contract authorization code is as follows:

```
function empower(address addr, string _name, string _useraddr, bool
_generaterights, bool _userights) public
{
    require(msg.sender == administartor);
    user[addr].name = _name;
    user[addr].useraddr = _useraddr;
    user[addr].generationPower = 0;
    user[addr].totalGenerationPower = 0;
    user[addr].usePower = 0;
    user[addr].totalUsePower = 0;
    user[addr].generaterights = _generaterights;
    user[addr].userights = _userights;
    user[addr].information = "";
}
```

The function starts to verify the permissions to ensure that the caller of the function is the contract publisher, and then the parameters are passed to the corresponding user structure. Addr is the unique Ethereum address of the user to be registered, which is written into each meter as the unique identification of the meter or user. Here, the administrator can grant the power generation right and power consumption right to the user through generaterights and Userrights member variables. The initial energy consumption is initialized to 0, and the dispatch information is initialized to null. These variables are controlled by various functions.

### 5.1.2 Implementation of related member function

There are two kinds of functions: one will change the value of the variable in the contract. All calls of this function will be recorded in the blockchain in the form of a transaction, which ensures the authenticity and traceability of the data. The other is the query function, which does not change the value of the variable in the contract, so it does not need to be recorded in the blockchain.

#### 5.1.2.1 Implementation of Generate Power function

After getting the power generation right, the power generation users can generate electricity according to the relevant regulations and plans, and upload the real-time generation data to the blockchain by calling the incremental generation function in the contract. The function is implemented as follows:

```
function GeneratePower(uint256 _value) public
{
    if(!user[msg.sender].generaterights)
        return;
    user[msg.sender].generationPower += (int256)(_value);
    user[msg.sender].totalGenerationPower += _value;
}
```

The function takes the generated capacity as a parameter and has no return value. The function first verifies the authority to make sure that only the authorized generation users can call the function. Then, the function adds the generated capacity of the caller to the total energy generation variable in the user structure and the current required generation variable. Since the variables in the contract are changed in the

process, the call will be made in a similar ratio. The form of token transactions is permanently recorded in the blockchain.

#### 5.1.2.2 Implementation of Transfer Power function

After the power generation right is obtained, the power generation users can sell the generated energy to the electricity users. Here, the contract realizes an electricity trading function. By calling this function, the generating users can realize the function of selling electricity. The specific implementation is as follows:

```
function transfer(address _to, uint256 _value) public
{
    require(user[msg.sender].generaterights);
    require(user[_to].userights);
    require(user[_to].usePower + _value > user[_to].usePower);
    user[msg.sender].generationPower = (int256)(_value);
    user[_to].usePower += _value;
}
```

The function has two input parameters. One is who the electricity will be sold to, and the other is the electricity sold. There is no return value. The caller of this function should be a generation user. Therefore, the authority verification is carried out first to verify whether the caller has the generation right. Then, it is necessary to verify whether the receiver has the useful power right, that is, whether it is a registered user. Finally, it is necessary to ensure that the electricity sold must be positive and negative electricity can not be sold. If all the above conditions are met, the generating user will subtract the corresponding electricity at the same time, the user increases the corresponding electricity as the electricity purchased. Because the variables in the contract are changed, the process will be permanently recorded in the blockchain in the form of a bitcoin transaction.

#### 5.1.2.3 Implementation of Consume Power function

After the power users purchase the electricity from the power generation users, they will have the available electricity. At the same time, the users can automatically call the power consumption function in real time to deduct the corresponding electricity from the user's account. The specific implementation of the function is as follows:

```

function PowerConsume(uint256 _value) public
{
    require (use[msg.sender].userights);
    require (user[msg.sender].usePower >= _value);
    user[msg.sender].usePower -= _value;
    user[msg.sender].totalUsePower += _value;
}

```

The function starts with two verifications, one is to verify whether the user has the right to use electricity, and the other is to verify whether the user has enough power. The input parameter of the function is the power consumed by the user in real time. The corresponding energy is subtracted from the corresponding user's account. Because the value of the internal variable of the contract is changed, this operation will be permanently recorded on the blockchain.

#### 5.1.2.4 Implementation of Dispatch Power function

According to the real-time operation of the power grid, the power manager can send dispatching information to each user. The specific code is as follows:

```

function PowerDispatch(address addr, string _info) public
{
    require(msg.sender == administrator);
    user[addr].information = _info;
}

```

The function has two parameters, one is the address information, which is to whom the scheduling information is published, and the other is the content of the scheduling information. The release of scheduling information must be released by the administrator, so authentication is carried out at the beginning of the function to ensure that the function caller is the system administrator. This function also changes the content of the contract variable, so it will be permanently recorded on the blockchain.

#### 5.1.2.5 Implementation of Query Information function

The contract reserves the following query interfaces, through which users can query the correctness of the data recorded on the blockchain and verify the correctness of the local database data.

Electricity users can query the remaining electricity in their current accounts to determine whether they need to buy electricity; at the same time, power users can also query how much electricity they have used in total. These two functions are implemented as follows:

```
function GetUsedOfSelf() public constant returns(uint256 _value)
{
    return (user[msg.sender].usePower);
}

function GetTotalUsedOfSomeone(address addr) public constant
returns(uint256 _value)
{
    if(msg.sender == addr || msg.sender == administarator)
        return (user[addr].totalUsePower);
}
```

View their own residual power function, no input parameters, only query the function caller's residual power, which ensures the privacy of users and avoids being used by criminals. To view the user's total electric energy function, there is an address variable as the input parameter, which represents the address of the user to be checked. However, at the beginning of the function, the function is limited to ensure that only the user or the administrator can view it, which also avoids the leakage of user information and being used by criminals.

Generation users can also query their current demand for power generation and the total amount of power generation. The specific implementation of the function is as follows:

```
function GetGeneration() public constant returns(int256 _value)
{
    return (user[msg.sender].generationPower);
}

function GetTotalGeneration(address addr) public constant
returns(uint256 _value)
{
    if(msg.sender == addr || msg.sender == administarator)
        return (user[addr].totalGenerationPower);
}
```

There are no input parameters for the function to view the current required generation, which ensures that only the caller's own current required generation can be queried. There is an address variable as the



input parameter for the function to view the total amount of power generation, and only users and administrators have permission.

In addition, both power users and power generation users may receive dispatching information. The function of viewing their own dispatching information is as follows:

```
function GetInformation()public constant returns(string _info)
{
    return (user[msg.sender].information);
}
```

The function also has no input parameters to ensure that only its own scheduling information can be viewed. The smart meter calls the function periodically. Once the dispatching information is read, the corresponding processing is made immediately according to the dispatching information.

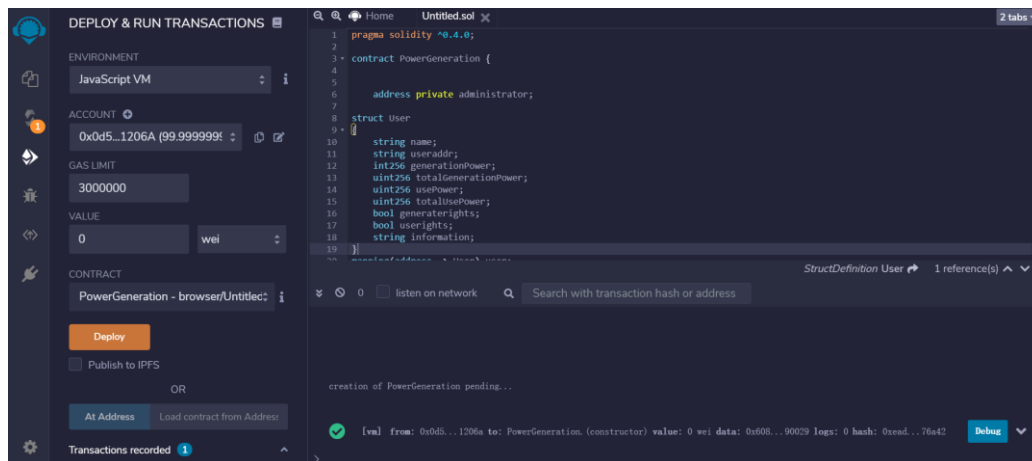
Consumption users and generation users can query data online at any time by calling the above functions.

## **5.2 Compilation and debugging of smart contract**

In this subsection we compile and debug our smart contract to ensure it can implement the functions that we need.

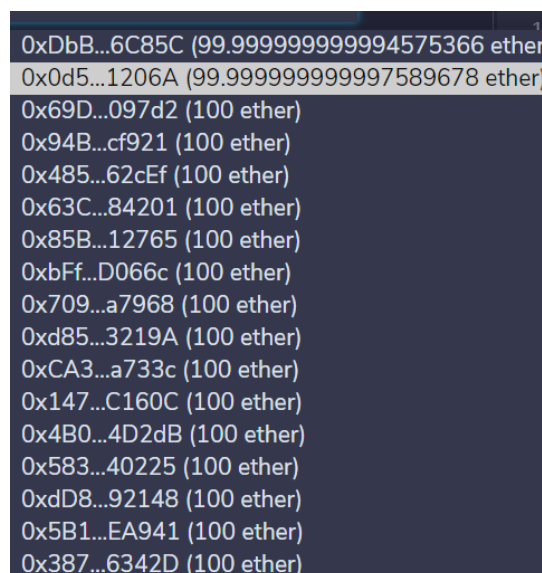
### **5.2.1 Compilation of smart contract**

After the contract code is written, the next step is to compile and debug. According to the working environment, set the running conditions and simulate the smart contract. The following operations are based on Remix. The running environment of it is the virtual machine provided by Ethereum. This is a virtual machine that simulates the blockchain in memory. Using this virtual machine, you can quickly debug the smart contract. Therefore, select JavaScript VM environment in Remix compiler: after selection, there are several virtual accounts provided by virtual machines. The first one is the contract deployment account by default. Put the contract code in the code editing area in the upper right area. After clicking the Deploy button, you can observe that the contract has been deployed to the virtual machine and all functions are ready. As shown in Figure 5-1



**Figure 5-1: Deployment of contract in Remix**

At the upper right of the compiler is the code area, which is the smart contract we wrote. At the bottom right is some blockchain transaction information. Every step of operation will be recorded. If the variable is changed, it will be permanently recorded in the blockchain in the form of transaction. If the variable value is not changed, an operation information will be generated to show the operation result. On the top left is the virtual account area, where the virtual machine provides several virtual Ethereum accounts to simulate the power users or power generation users in our real environment. The account addresses are can be seen in Figure 5-2:



**Figure 5-2: Ethereum accounts provided by Remix**

The first account is the account that publishes the smart contract by default, that is, the account of the power management agency as we call it. We will not make any changes here. The remaining accounts are ordinary users, which will be set up as electricity users and power generation users later. At the bottom left is the function interface area in the contract. We can use these functions to realize the call of smart contract to simulate the call process of smart grid platform to the contract in reality. As you can see in Figure 5-3, the functions are divided into blue and red. These are the functions that do not change the variables in the contract and the functions that change the variables in the contract.

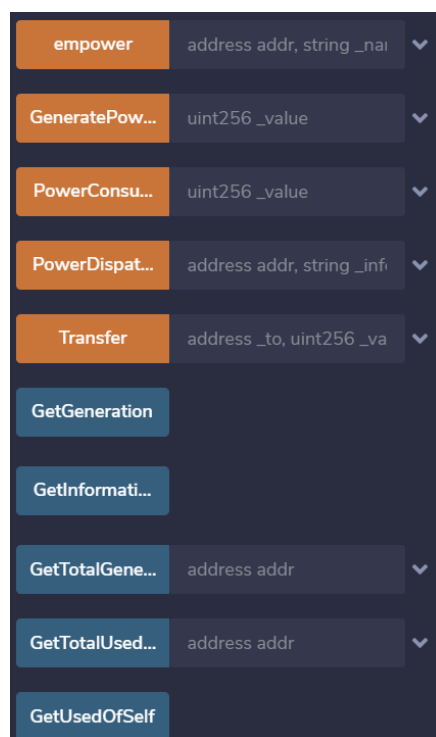


Figure 5-3: functions of smart contract

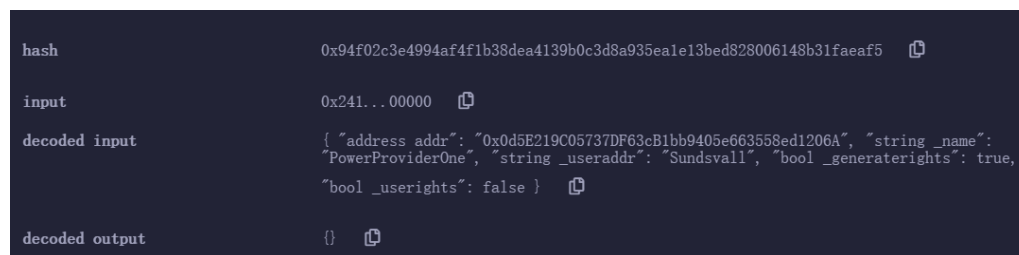
### 5.2.2 Debugging and running of smart contract

After the contract is deployed, the constructor is executed automatically. The constructor initializes the contract deployer as the administrator. Both the power generation user and the power user need to apply to the administrator for registration. In the initialization state, we call any function will not change the variables in the contract, because no user gets the power generation or power consumption right. Let's set up the second and third accounts as generation users and the fourth and fifth

accounts as consumers. According to the function requirements, we need to provide the following information as authorization information. The information to be provided includes: the user's unique address, that is, Ethereum address, user name, user address, that is, the user's location. These are used to limit a unique node. In addition, the user's current required power generation, cumulative power generation, current remaining power and cumulative power consumption are initialized. For example, if the second user is set as the generation user, the input parameter format is as follows:

```
"0x0d5E219C05737DF63cB1bb9405e663558ed1206A","PowerProviderOne", "Sundsvall",true,false
```

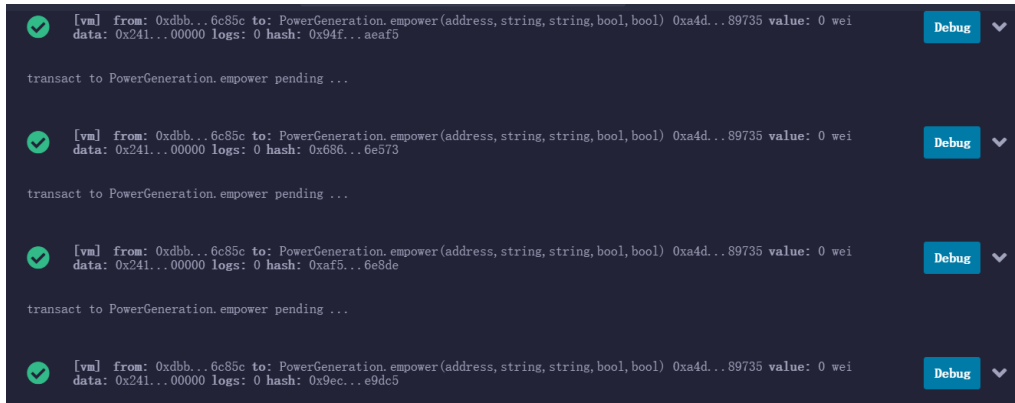
Click the red button next to it to see a transaction information at the bottom left. Click open to see that the information we set has been written into the blockchain, as shown in Figure 5-4.



hash	0x94f02c3e4994af4f1b38dea4139b0c3d8a935ea1e13bed828006148b31faeaf5
input	0x241...0000
decoded input	{ "address addr": "0x0d5E219C05737DF63cB1bb9405e663558ed1206A", "string _name": "PowerProviderOne", "string _useraddr": "Sundsvall", "bool _generaterights": true, "bool _userights": false }
decoded output	{}

**Figure 5-4: Transaction information written into the blockchain**

The first line is the hash of this transaction, which can uniquely represent the transaction for future query; the second line is the input of Ethereum contract ABI format of this function; the third line is the information in JSON format, where the user name and address are in ASCII code format; the fourth line is output, where you can see that this function has no output. Similarly, we initialize other accounts, and each operation will be permanently recorded in the blockchain, and the details can be viewed. The transaction information area is shown in Figure 5-5. We can see that the first contract deployment and the following four administrator authorization operations are recorded.



**Figure 5-5: Transaction information area**

After initialization, let's simulate electricity trading. First of all, as a power user, after obtaining the power consumption right, the user's available electricity is 0. At this time, the power user can purchase the electricity from the generation user. The purchase function is the transfer function, which is called by the generation user. The user's address and quantity of electricity purchased are taken as parameters, and there is no return value. Here, we assume that generation user 1 sells 100 units of electricity to user 1, and is selling before electricity supply, we can check the current demand of power generation user 1 and the remaining electricity quantity of user 1. Switch to the account of generation user 1 in the virtual machine area, and simulate that the caller is generation user 1. Click getgeneration function to get the result as shown in following. Switch to power user 1 and click getuseofself function to get the result as shown in following

```
0: int256: _value 0
```

```
0: uint256: _value 0
```

It can be seen that the function returns 0, which means that power generation user 1 has not sold electric energy, and the current remaining energy of power consumer 1 is 0

Now the generation user sells 100 units of energy to consumer 1 and calls the electricity trading function. Switch to generation user 1, click the transfer function, and you can see that this transaction appears in the trading area at the bottom left. Click to view the details, and you can see

the hash and other information of this transaction, as shown in Figure 5-6.

```
input          0x69c...00064  ⓘ
decoded input  { "address _to": "0x69DEcf193967f83609eb67632b426ef1587097d2", "uint256 _value":
               "_hex": "0x64" } } ⓘ
decoded output  {} ⓘ
```

**Figure 5-6: Transaction information of this operation**

Once again, check the current required generation capacity of generation user 1 and the remaining electricity consumption of power consumer 1, and the results are shown in following.

```
0: int256: _value -100

0: uint256: _value 100
```

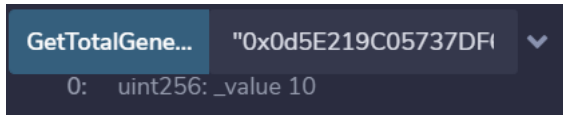
It can be seen that the current required generation capacity of power generation user 1 becomes -100, which means that there are 100 units of electricity to be generated at present, and the remaining power of user 1 is 100, which means that the remaining power in the current account is 100 units.

This generation user 1 and user 1 complete an electricity transaction. When generating electricity, generating user 1 calls the function of generating electricity to record the added electricity in real time, while user 1 uses electricity, it calls the power consumption function to record the electricity used in real time. Switch to power generation user 1, call the power generation function, and the parameter is real-time generation. Here, assume that the power generation is 10 units. Fill in the parameters. After clicking the GeneratePower function, you can see that a transaction record is generated in the trading area. At this time, check your current required power generation and total power generation again, and you can see the result as shown in following.

```
0: int256: _value -90
```

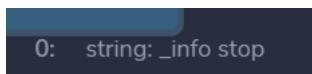
The current required power generation is 90. To view the total generation function, you need to input the user address as the input parameter. Here is the address of generation user 1. After calling, the function returns the

total generation of this account to 10, which is exactly the 10 units of electricity we just sent. As shown in following.



In addition, the power management organization can call the function of dispatching information release to send dispatching information to users. The parameters of dispatching information function need a user address, that is, to whom dispatching information is sent, and the other is the content of dispatching information. Here, it is assumed that the manager needs to send a dispatch order with the content of "stop" to power user 1

Order. After switching to the administrator account, click the function to see a transaction information in the trading area at the bottom left. At this time, switch to power generation user 1 to view their own dispatching information: you can see the dispatching information as shown in following.



This section tests all functions of the contract, and each function has achieved the expected effect.

### 5.3 Build a private chain

In this section, geth client will be used to build the Ethereum blockchain private chain, and the construction process is shown in Figure 5-7.

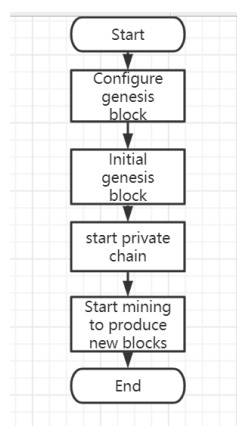


Figure 5-7: Construction process of Ethereum private chain

As mentioned above, the genesis block of the block chain is generated through parameter configuration. Ethereum supports custom creation block, and the creation block information is written in a JSON format configuration file. For example, genesis.json contains the following contents:

```
{
  "config": {
    "chainId": 666,
    "homesteadBlock": 0,
    "eip150Block": 0,
    "eip150Hash": "0x0000000000000000000000000000000000000000000000000000000000000000",
    "eip155Block": 0,
    "eip158Block": 0,
    "byzantiumBlock": 0,
    "constantinopleBlock": 0,
    "petersburgBlock": 0,
    "istanbulBlock": 0,
    "ethash": {}
  },
  "nonce": "0x0",
  "timestamp": "0x5ddf8f3e",
  "extraData": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "gasLimit": "0x47b760",
  "difficulty": "0x00002",
  "mixHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "coinbase": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "alloc": { },
  "number": "0x0",
  "gasUsed": "0x0",
  "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000"
}
```

Once the genesis block is configured, the private chain can be started from the command line. Initialize the private chain by entering the following command:

```
C:\Users\62955>d:
D:\>cd D:\Software\geth
D:\Software\geth>geth --datadir "$basepath/chain" init genesis.json_
```

And you can get response as following:



```
D:\Software\geth>geth --datadir "$basepath/chain" init genesis.json
INFO [09-14|17:30:41.878] Maximum peer count                ETH=50 LES=0 total=50
INFO [09-14|17:30:42.107] Set global gas cap                 cap=25000000
INFO [09-14|17:30:42.122] Allocated cache and file handles   database=D:\chain\geth\chaindata cache=16.00MiB handles=16
INFO [09-14|17:30:42.149] Writing custom genesis block
INFO [09-14|17:30:42.154] Persisted trie from memory database nodes=0 size=0.00B time=0s gcnodes=0 gcspace=0.00B gctrans=0s livenodes=1 livesize=0.00B
INFO [09-14|17:30:42.162] Successfully wrote genesis state    database=chaindata hash="d3d6bb...c5304a"
INFO [09-14|17:30:42.165] Allocated cache and file handles   database=D:\chain\geth\lightchaindata cache=16.00MiB handles=16
INFO [09-14|17:30:42.182] Writing custom genesis block
INFO [09-14|17:30:42.184] Persisted trie from memory database nodes=0 size=0.00B time=0s gcnodes=0 gcspace=0.00B gctrans=0s livenodes=1 livesize=0.00B
INFO [09-14|17:30:42.192] Successfully wrote genesis state    database=lightchaindata hash="d3d6bb...c5304a"
```

After initialization, the private node can be started. Enter the following command:

```
D:\Software\geth>geth --rpc --rpccorsdomain "*" --datadir "$basepath/chain" --port "30303" --rpcapi "db,eth,net,web3" --networkid 123 --nodiscover console
```

After startup, the following information indicates that the Ethereum private node was successfully started.

```
Welcome to the Geth JavaScript console!

instance: Geth/v1.9.21-stable-0287d548/windows-amd64/gol.15
at block: 0 (Thu Nov 28 2019 17:11:26 GMT+0800 (CST))
datadir: D:\chain
modules: admin:1.0 debug:1.0 eth:1.0 ethash:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0
```

Type the *eth.BlockNumber* command on the command line, query the current blockNumber, and see the return of 0, indicating that there are currently 0 blocks.

```
> eth.blockNumber
0
```

This is because we don't have an account yet, so we need to create an account and execute the following command:

```
> personal.newAccount("password")
```

And we get a Ethereum account as following.

```
INFO [09-14|19:17:44.317] Your new key was generated          address=0xCd40f4950A76718bCd90511ae2Ba0171F4b0ba52
WARN [09-14|19:17:44.321] Please backup your key file!        path=D:\chain\keystoreUTC--2020-09-14T11-17-41.547626100Z--cd40f4950a76718bcd90511ae2ba0171f4b0ba52
WARN [09-14|19:17:44.326] Please remember your password!      "0xcd40f4950a76718bcd90511ae2ba0171f4b0ba52"
```

With this account we can perform mining with method *miner.start()*. After a period of time, Query the number of blocks again with *eth.blockNumber*. It can be seen that:

```
> miner.start()  
null  
> eth.blockNumber  
18  
>
```

So far, the Ethereum private chain has been built and started successfully, and the next step will be to deploy the intelligent contract we wrote before into the blockchain.

## 5.4 Deploy smart contract

There are many ways to deploy an intelligent contract. Here we use the Truffle framework, and the basic process of using it for development is shown in Figure 5-8:

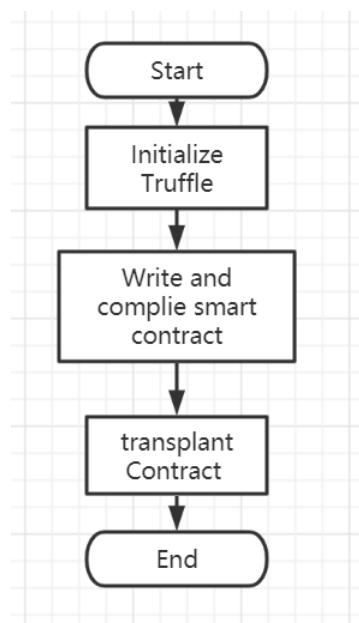
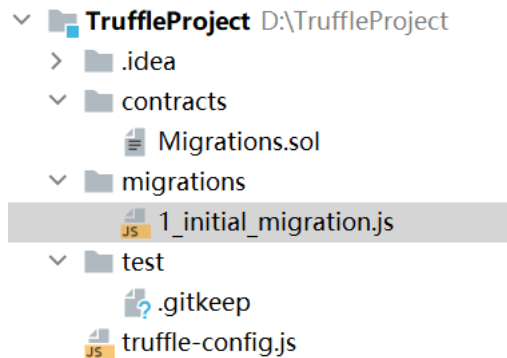


Figure 5-8: Deployment process of the smart contract

### 5.4.1 Initialize truffle

First create a directory wherever you want to put your project, and then initialize the truffle project with *truffle init* command. When done, you get the directory shown in Figure 5-9:



**Figure 5-9: Directory of the Truffle project**

The TruffleProject directory is the default directory for running your application files. This includes recommended directories for JavaScript files and CSS files, but you can fully decide how to use them. You can write your own JavaScript files and CSS files into the corresponding directory, replace the original file, to build your own application. The Contracts directory stores the directory for Truffle default contract files. Migrations is a contract migration directory, which makes the Truffle framework a migration feature. The Test directory is used to Test files. *truffle -config.js* has the basic compilation path configured for the truffle configuration file, as well as the blockchain listening port.

#### 5.4.2 Compile and migrate smart contract

We've created a default application with *truffle init*, and we need to modify the files to create our own project. Here we put the contract we wrote, *Powergeneration.sol*, into the contracts folder and keep the *migrations.sol* file, which truffle uses to help deploy the contract.

Then modify the *1\_initial\_migration.js* file in the migrations directory, and the content is as follows:

```
const Migrations = artifacts.require("Migrations");  
  
module.exports = function (deployer) {  
  deployer.deploy(PowerGeneration);  
};
```

Then enter *truffle compile* command to compile our smart contract.

```
$ truffle compile
Compiling Migrations.sol...
Compiling PowerGeneration.sol...
Writing artifacts to ./build/contracts
```

Start the Ethereum private chain established in the previous section, and start mining. Enter the following command to migrate the smart contract. When the command line sees the following output, it indicates that the contract has been successfully deployed into the Ethereum private chain created in the previous section.

```
$ truffle migrate
Running migration: 1_initial_migration.js
  Deploying Migrations...
  Migrations: 0x42f423526381d685261b883e68902ee9a1705ec5
Saving successful migration to network...
Saving artifacts...
Running migration: 2_deploy_contracts.js
  Deploying PowerGeneration...
  PowerGeneration: 0x4e2e411514015d419b4f389be5bb2c24cb194778
Saving successful migration to network...
Saving artifacts...
```

## 5.5 Summary of this chapter

In this chapter, the intelligent contract is first written. Smart contract is the most core part of smart grid system design, which realizes the real-time storage of user power data into the block chain, ensuring the authenticity and traceability of data. The contract also provides a power transaction function, which allows the generator to sell electricity to the user. In addition, as a power management agency, the manager can adjust the power grid operation in real time according to the real-time operation of the power grid by releasing the dispatching information through the dispatching information publishing function. Users can view their own scheduling information by scheduling information view functions. After the contract was written, the ethereum private chain was built by modifying the creation file. Then, using the Truffle framework, the smart contract is deployed into the Ethereum private chain.

## 6 Results

This chapter will test the performance of the system by testing the response time of the system under different loads. The test content includes testing the response time of a fixed number of users to realize various functions under different number of blocks, and testing the response time of different number of users to realize various functions under a fixed number of blocks. The testing tool is LoadRunner. The test environment is a personal computer, the CPU is intel i5-10210u @1.6GHz, the RAM is 32GB, and the operating system is windows 10.

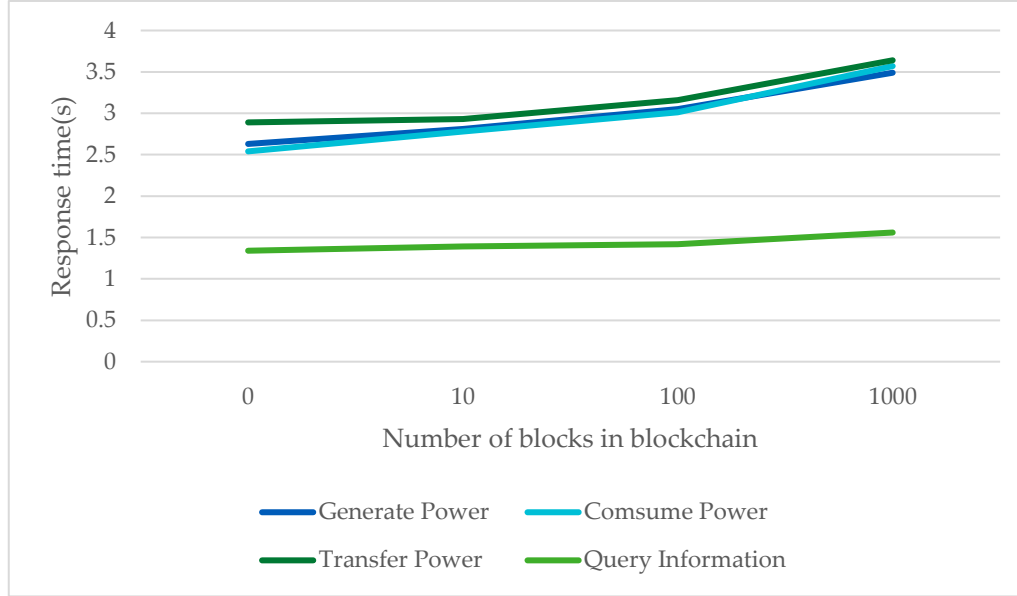
### 6.1 Fixed number of users, different number of blocks

First we test the response time of a fixed number of users to realize various functions under different number of blocks. The number of users is 100. Each function is tested 100 times. Table 6.1 shows the average and standard deviation of the response time taken by a user to implement various functions of the system.

Function	Number of blocks in blockchain			
	0	10	100	1000
Generate Power	$\mu=2.63$ $\sigma=0.01$	$\mu=2.81$ $\sigma=0.04$	$\mu=3.05$ $\sigma=0.12$	$\mu=3.49$ $\sigma=0.47$
Consume Power	$\mu=2.54$ $\sigma=0.03$	$\mu=2.78$ $\sigma=0.06$	$\mu=3.01$ $\sigma=0.10$	$\mu=3.57$ $\sigma=0.24$
Transfer Power	$\mu=2.89$ $\sigma=0.10$	$\mu=2.93$ $\sigma=0.06$	$\mu=3.16$ $\sigma=0.18$	$\mu=3.64$ $\sigma=0.23$
Query Information	$\mu=1.34$ $\sigma=0.02$	$\mu=1.39$ $\sigma=0.04$	$\mu=1.42$ $\sigma=0.03$	$\mu=1.56$ $\sigma=0.09$

**Table 6.1: response time of a fixed number of users to realize various functions under different number of blocks**

Figure 6.1 shows the content in Table 6.1. Its function is to visualize the trend of system response time with the number of blocks.



**Figure 6.1: response time of a fixed number of users to realize various functions under different number of blocks**

## 6.2 Different number of users, fixed number of blocks

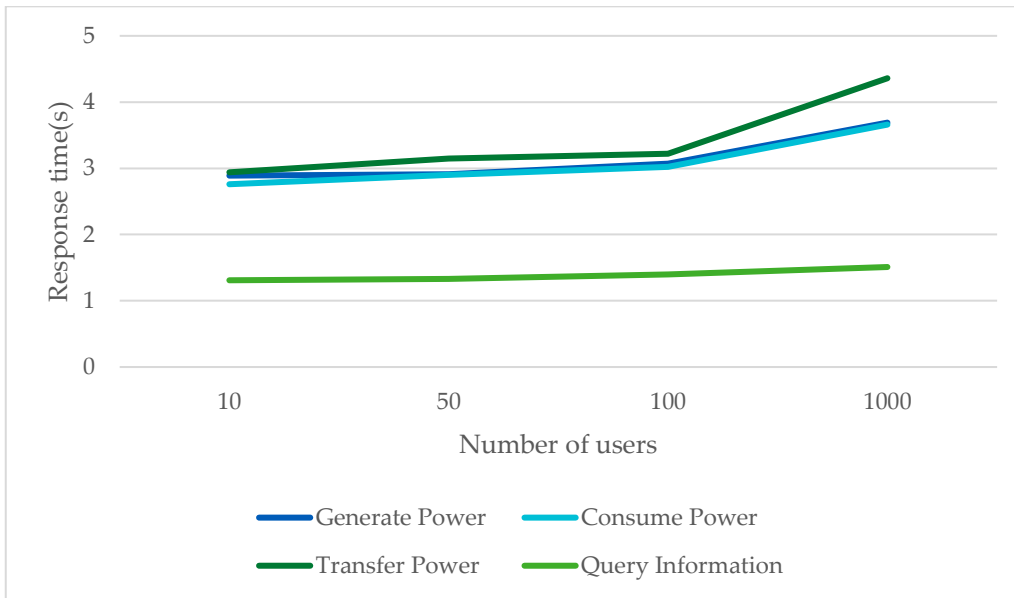
Now we test the response time of different numbers of users to realize the various functions of the system under the fixed number of blocks. The number of blocks is 100. Table 6.2 shows the response time taken by the system to implement various functions under different numbers of users.

Function	Number of users in system			
	10	50	100	1000
Generate Power	$\mu=2.89$ $\sigma=0.04$	$\mu=2.91$ $\sigma=0.06$	$\mu=3.07$ $\sigma=0.09$	$\mu=3.69$ $\sigma=0.27$
Consume Power	$\mu=2.76$ $\sigma=0.03$	$\mu=2.90$ $\sigma=0.04$	$\mu=3.02$ $\sigma=0.08$	$\mu=3.66$ $\sigma=0.23$

Transfer Power	$\mu=2.94$ $\sigma=0.10$	$\mu=3.15$ $\sigma=0.08$	$\mu=3.22$ $\sigma=0.12$	$\mu=4.36$ $\sigma=0.23$
Query Information	$\mu=1.31$ $\sigma=0.01$	$\mu=1.33$ $\sigma=0.03$	$\mu=1.40$ $\sigma=0.03$	$\mu=1.51$ $\sigma=0.05$

**Table 6.2: response time of a fixed number of blocks to realize various functions under different number of users**

Figure 6.2 shows the content in Table 6.2. Its function is to visualize the trend of system response time with the number of users.



**Figure 6.2: response time of a fixed number of blocks to realize various functions under different number of users**

### 6.3 Analysis of results

As we have seen in Chapter 6.1, when the number of users in the system is constant, the response time taken by the system to implement various functions is positively related to the number of blocks. But overall, the system response time is less affected by the number of blocks.

In chapter 6.2, when the number of blocks in the system is constant, the response time taken by the system to implement various functions is positively related to the number of users. Among them, Transfer power

function is a bit more greatly affected by the increase in the number of users. This may be because the function needs to find a suitable power generator, and the increase in the number of users affects the time spent in this process.



## 7 Conclusions

Through the research carried out in the paper, we have achieved the research goals we proposed. First of all, through in-depth study of blockchain technology, we understood the underlying principles of blockchain. Then we wrote a smart contract based on functions of the system and checked its correctness.

Next we created a smart grid system based on the private chain of Ethereum. Through this system, users can not only serve as power generators to provide power to other users in the system, but also as power consumers to purchase power from other users in the system. And all activities including power generation, power consumption and power transactions, will be recorded in the blockchain in the form of new blocks. The system guarantees the validity and non-repudiation of these actions through the blockchain.

Finally, we evaluated the performance of the system and proposed the future work. We test the performance of the system by testing the response time of the system under different loads. Through the method of controlling variables, we tested the response time of the system to achieve various functions under different number of blocks and different numbers of users. Through the change curve of response time with the change of independent variables, we conclude that the system response time is positively correlated with the number of blocks and the number of users. In other words, system performance is negatively correlated to them.

Blockchain technology is actually the integration of existing technologies, including hash algorithm, asymmetric encryption algorithm and so on. In this paper, through the preparation of smart contracts to achieve intelligent metering and automatic trading of smart grid, and using the framework of truffle to build a distributed application platform, users can view their own data stored in the blockchain through the platform, and also realize the power transaction through the platform. Each user can store all transaction records to ensure that the data is truly reliable. To build a management platform for the power management organization, the platform has no right to interfere with the user's right to sell or buy power, but it can issue dispatching instructions to users

according to the real-time operation of the power grid, so as to regulate the operation of the power grid intelligently.

## **7.1 Ethical and social aspects**

The smart grid platform based on the blockchain technology makes the user data more secure. The characteristics of the blockchain to trust make these user data even if transmitted in the public network, it can also ensure the data security, and save the high management costs caused by the centralized management of data and the use of dedicated line for transmitting data.

At the same time, the smart grid system, through the immutability of the blockchain, solves the integrity problem in the electric energy transaction process to a certain extent. Because any transaction in the smart grid system will be saved in the blockchain of all users in the form of new blocks. Therefore, any attempt to tamper with transaction records will not receive the approval of other users in the system.

## **7.2 Future work**

There are some shortcomings in this paper: (1) the preparation process of smart contract is very complex in the actual process, because once it is due to the vulnerability of smart contract, the consequences will be unimaginable. In reality, there are also examples of serious consequences caused by contract writing loopholes. Therefore, the smart contract of this paper needs to be carefully compiled under the background of fully understanding the power trading. (2) The functions of distributed application platform and management platform need to be improved. The platform should make it more convenient for users to use. It can realize data query and other functions by adding mobile app.

Thus in the next step of research work, the first step is to continue to improve the smart contract, try to avoid unnecessary losses due to loopholes. In addition, in the way of providing interaction with smart contracts, provide mobile access. Finally, we should further improve the system in combination with the actual operation.

# References

Below is an example of an automatically numbered list of references according to the numbered list and cross references method, as described in chapters 2.4:

- [1] Energimyndigheten. [n. d.]. Solceller.  
<http://www.energimyndigheten.se/fornymbart/solenergi/solceller>  
accessed 2020-10-26.
- [2] Bil Sweden. [n. d.]. Definitiva nyregistreringar under 2016.  
<http://www.bilsweden.se/statistik>  
accessed 2020-10-26.
- [3] V Cagri Gungor, Dilan Sahin, Taskin Kocak, Salih Ergut, Concettina Buccella, Carlo Cecati, and Gerhard P Hancke. 2013. A survey on smart grid potential applications and communication requirements. IEEE Transactions on industrial informatics 9, 1 (2013), 28–42.
- [4] Xi Fang, Satyajayant Misra, Guoliang Xue, and Dejun Yang. 2012. Smart grid—The new and improved power grid: A survey. IEEE communications surveys & tutorials 14, 4 (2012), 944–980.
- [5] Swan M. Blockchain: Blueprint for a new economy[M]. " O'Reilly Media, Inc.", 2015.
- [6] Turkanović M, Hölbl M, Košič K, et al. EduCTX: A blockchain-based higher education credit platform[J]. IEEE access, 2018, 6: 5112-5127.
- [7] Nakamoto S.Bitcoin:A peer-to peer electronic cash system[J].Consulted,2008
- [8] Blockchain: Everything You Need to Know  
<https://www.investopedia.com/terms/b/blockchain.asp>  
accessed 2020-10-26.
- [9] Block Header | Binance Academy  
<https://academy.binance.com/glossary/block-header>  
accessed 2020-10-26.

- [10] <https://www.cryptocompare.com/coins/guides/what-is-a-block-header-in-bitcoin/>  
Microsoft Corporation, 1992: Microsoft Word Användarhandbok. Ordbehandlingsprogram för Macintosh Version 5.0.  
accessed 2020-10-26.
- [11] Shi E. Analysis of deterministic longest-chain protocols[C]//2019 IEEE 32nd Computer Security Foundations Symposium (CSF). IEEE, 2019: 122-12213..
- [12] Chen Y, Xie H, Lv K, et al. DEPLEST: A blockchain-based privacy-preserving distributed database toward user behaviors in social networks[J]. Information Sciences, 2019, 501: 100-117.
- [13] Merkle R C. A certified digital signature[C]//Conference on the Theory and Application of Cryptology. Springer, New York NY, 1989: 218-238.
- [14] Merkle Tree - Wikipedia  
[https://en.wikipedia.org/wiki/Merkle\\_tree](https://en.wikipedia.org/wiki/Merkle_tree)  
accessed 2020-10-26.
- [15] Clack C D, Bakshi V A, Braine L. Smart contract templates: foundations, design landscape and research directions[J]. arXiv preprint arXiv:1608.00771, 2016.
- [16] What is Ethereum?  
<https://ethereum.org/en/what-is-ethereum/>  
accessed 2020-10-26.
- [17] Smart grids: what is a smart electrical grid – electricity networks in evolution  
<https://www.i-scoop.eu/industry-4-0/smart-grids-electrical-grid/>  
accessed 2020-10-26.
- [18] Smart Grid - Wikipedia  
[https://en.wikipedia.org/wiki/Smart\\_grid#Historical\\_development\\_of\\_the\\_electricity\\_grid](https://en.wikipedia.org/wiki/Smart_grid#Historical_development_of_the_electricity_grid)  
accessed 2020-10-26.

- [19] Khalid M, Savkin A V. Optimization and control of a distributed battery energy storage system for wind power smoothing[C]//2011 19th Mediterranean Conference on Control & Automation (MED). IEEE, 2011: 39-43.
- [20] Wang X. Collisions for hash functions MD4s MD5, NAVAL-128 and RIPEMD[J].Cryptology Eprint Archive deport, 2004, 2004.

# Appendix A: Source Code

## Smart Contract Code:

```
pragma solidity ^0.4.0;
contract PowerGeneration {
    address private administrator;
    struct User
    {
        string name;
        string useraddr;
        int256 generationPower;
        uint256 totalGenerationPower;
        uint256 usePower;
        uint256 totalUsePower;
        bool generaterights;
        bool userights;
        string information;
    }
    mapping(address => User) user;
    function PowerGeneration() public
    {
        administrator = msg.sender;
    }
    function empower(address addr, string _name, string _useraddr, bool
    _generaterights, bool _userights) public
    {
        require(msg.sender == administrator);
        user[addr].name = _name;
        user[addr].useraddr = _useraddr;
        user[addr].generationPower = 0;
        user[addr].totalGenerationPower = 0;
        user[addr].usePower = 0;
        user[addr].totalUsePower = 0;
        user[addr].generaterights = _generaterights;
        user[addr].userights = _userights;
        user[addr].information = "";
    }
}
```

```

function GeneratePower(uint256 _value) public
{
    if(!user[msg.sender].generaterights)
        return;
    user[msg.sender].generationPower += (int256)(_value);
    user[msg.sender].totalGenerationPower += _value;
}
function Transfer(address _to, uint256 _value) public
{
    require(user[msg.sender].generaterights);
    require(user[_to].userights);
    require(user[_to].usePower + _value > user[_to].usePower);
    user[msg.sender].generationPower -= (int256)(_value);
    user[_to].usePower += _value;
}
function PowerConsume(uint256 _value) public
{
    require(user[msg.sender].userights);
    require(user[msg.sender].usePower >= _value);
    user[msg.sender].usePower -= _value;
    user[msg.sender].totalUsePower += _value;
}
function PowerDispatch(address addr, string _info) public
{
    require(msg.sender == administrator);
    user[addr].information = _info;
}
function GetUsedOfSelf() public constant returns(uint256 _value)
{
    return (user[msg.sender].usePower);
}
function GetTotalUsedOfSomeone(address addr) public constant
returns(uint256 _value)
{
    if(msg.sender == addr || msg.sender == administrator)
        return (user[addr].totalUsePower);
}
function GetGeneration() public constant returns(int256 _value)
{

```

```

        return (user[msg.sender].generationPower);
    }
    function GetTotalGeneration(address addr) public constant
    returns(uint256 _value)
    {
        if(msg.sender == addr || msg.sender == administrator)
            return (user[addr].totalGenerationPower);
    }
    function GetInformation()public constant returns(string _info)
    {
        return (user[msg.sender].information);
    }
}

```

#### **Genesis.json code:**

```

{
  "config": {
    "chainId": 666,
    "homesteadBlock": 0,
    "eip150Block": 0,
    "eip150Hash":
    "0x0000000000000000000000000000000000000000000000000000000000000000",
    "eip155Block": 0,
    "eip158Block": 0,
    "byzantiumBlock": 0,
    "constantinopleBlock": 0,
    "petersburgBlock": 0,
    "istanbulBlock": 0,
    "ethash": {}
  },
  "nonce": "0x0",
  "timestamp": "0x5ddf8f3e",
  "extraData":
  "0x0000000000000000000000000000000000000000000000000000000000000000",
  "gasLimit": "0x47b760",

```



```
"difficulty": "0x00002",
"mixHash":
"0x0000000000000000000000000000000000000000000000000000000000000000
000",
"coinbase": "0x000000000000000000000000000000000000000000000000",
"alloc": { },
"number": "0x0",
"gasUsed": "0x0",
"parentHash":
"0x0000000000000000000000000000000000000000000000000000000000000000
000"
}
```